Lab 8: Debugging in VSCode/Venus

Taaruni Ananya | Session 008 | CRN: 15623

02/28/2025

Dr. Lan Gao

Introduction

Lab 8 is centered around debugging which involves practicing writing code with RISC-V, SNOWBALL, and VSCode as well as learning how to use debugging tools. Lab 8 looks into exploring how to use error messages and values to fix programs. The lab explains how to use the terminal's error messaging system, breakpoints, and values. The lab also gives hands-on experience with fixing code with errors that may not be picked up by the assembler..

Purpose

The purpose of this lab is to learn how to debug efficiently as well as continue practicing the use of Assembly code, along with the commands and practice writing RISC-V utilizing VSCode and Terminal; this lab also works to introduce errors and their structures to fix syntax and code mistakes. This lab is meant to experiment with using terminal to identify issues with programs and explore different ways to address them.

Main Objectives

- Learn several different errors that assemblers may point out
- Learn what problems are being highlighted by the assembler
- Learn how to address and fix such errors
- Learn that assemblers are not capable of catching all errors, and that programs can assemble even with certain mistakes
- Learn about “gdb” programs for debugging executable programs
- Perform add, subtract, NOT, NEG operations
- Learn how to set breakpoints and step through programs
- Learn how to retain information about registers, PC, etc.

Brief Explanation

VS Code is an integrated development environment (IDE) where Venus and RISC-V can be accessed and programmed. “Terminal is an app for advanced users and developers that lets you communicate with the Mac operating system using a command line interface (CLI).” (Apple). Using these two softwares and the assembler, a programmer can write Assembly code and catch errors to e=debug efficiently. Terminal provides a place to run programs that can output errors, what’s causing them, what line, and which column to better assist programmers.

Methodology/Procedure

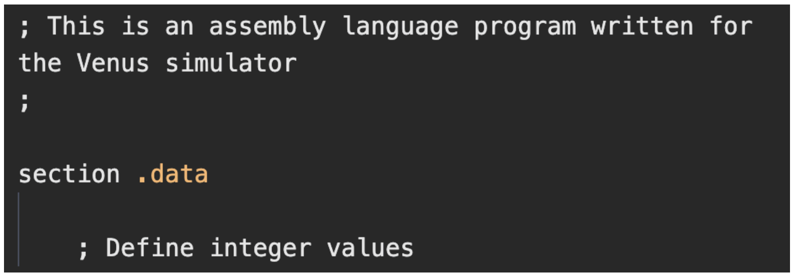
The procedure starts with learning using a mistake-filled code provided in the slideshow. The several different errors in the code are explored with cTerminal. Then, each error is addressed, found the root cause of, fixed, and moved onto the next error. Next, “gdb” is explored to understand and to debug executable programs. Registers, PC, and breakpoints are next explored to further understand and deepen knowledge of efficient debugging tools, systems, and procedures.

Documentation

Part one:

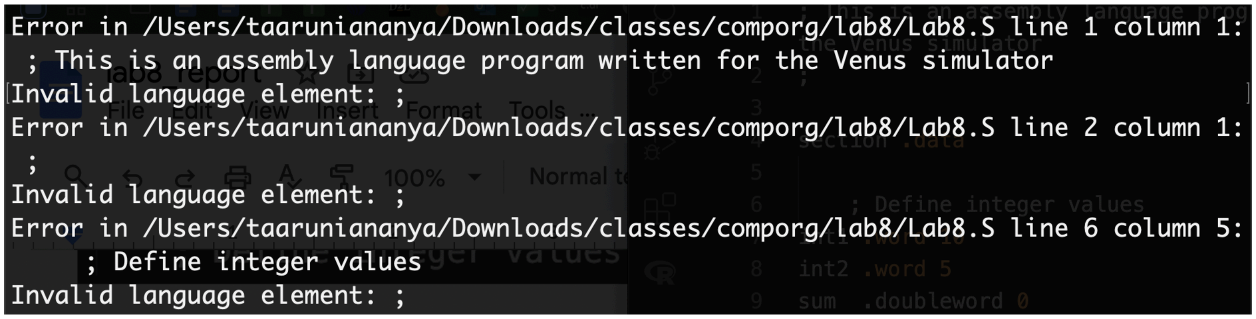
Error One (;):

Original Code:



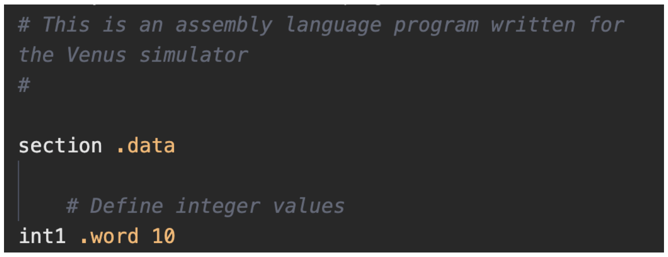
```
; This is an assembly language program written for
the Venus simulator
;

section .data

    ; Define integer values
```

Error statement:



```
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 1 column 1:
 ; This is an assembly language program written for the Venus simulator
Invalid language element: ;
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 2 column 1:
 ;
Invalid language element: ;
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 6 column 5:
    ; Define integer values
Invalid language element: ;
```

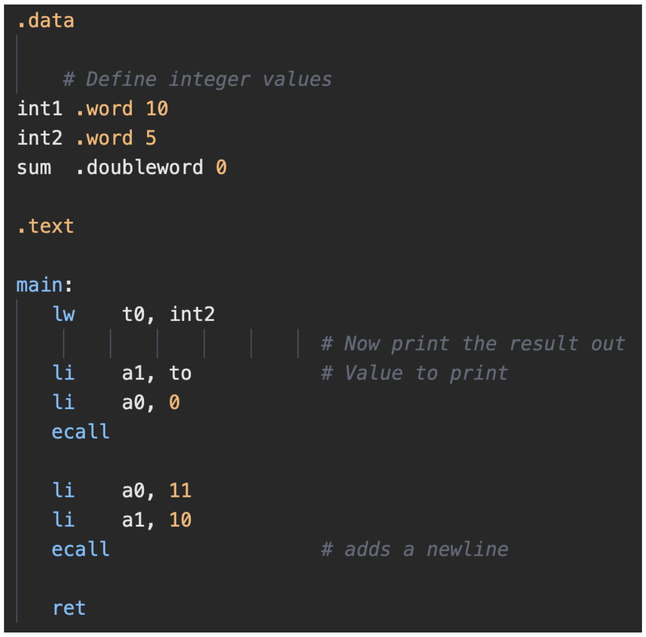
Fixed Code:



```
# This is an assembly language program written for
the Venus simulator
#

section .data

    # Define integer values
int1 .word 10
```

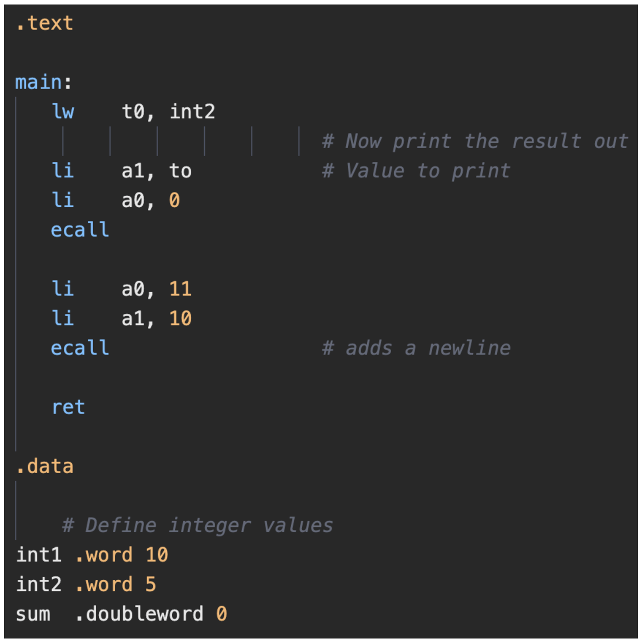
Error Two (.data):

Original code:



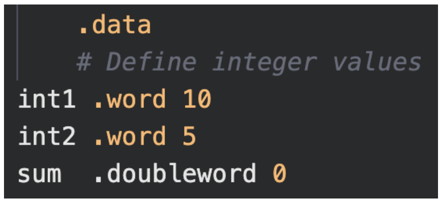
```
.data

    # Define integer values
int1 .word 10
int2 .word 5
sum  .doubleword 0

.text

main:
   lw    t0, int2
                            # Now print the result out
   li    a1, to             # Value to print
   li    a0, 0
   ecall

   li    a0, 11
   li    a1, 10
   ecall                    # adds a newline

   ret
```

Changed code:

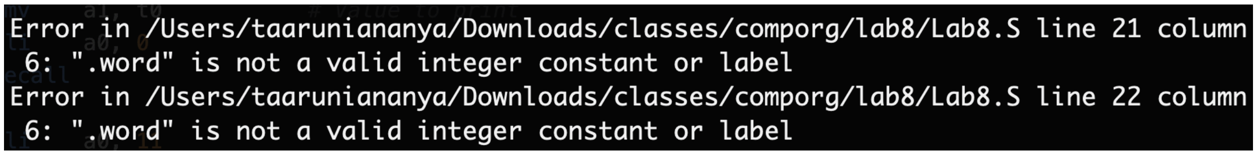

```
.text

main:
    lw    t0, int2
                          # Now print the result out
    li    a1, to          # Value to print
    li    a0, 0
    ecall

    li    a0, 11
    li    a1, 10
    ecall                 # adds a newline

    ret

.data

    # Define integer values
int1 .word 10
int2 .word 5
sum  .doubleword 0
```

Error Three (.word):

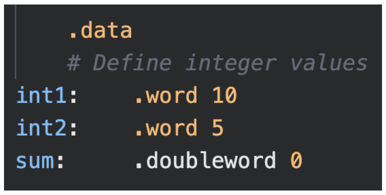
Original code:


```
     .data
     # Define integer values
int1 .word 10
int2 .word 5
sum  .doubleword 0
```

Error:

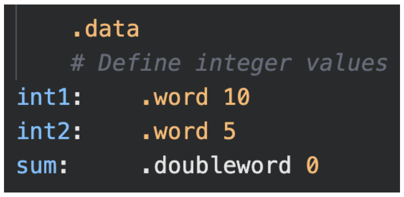

```
Error in /Users/taarunianaya/Downloads/classes/comporg/lab8/Lab8.S line 21 column
 6: ".word" is not a valid integer constant or label
Error in /Users/taarunianaya/Downloads/classes/comporg/lab8/Lab8.S line 22 column
 6: ".word" is not a valid integer constant or label
```

Fixed Code:


```
    .data
    # Define integer values
int1:     .word 10
int2:     .word 5
sum:      .doubleword 0
```

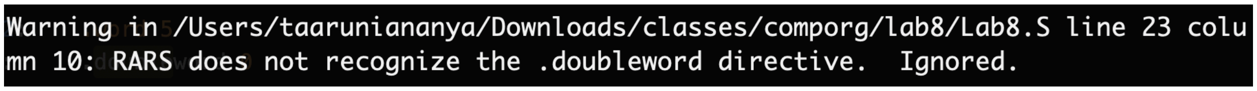
Error Four (.doubleword):

Original Code:



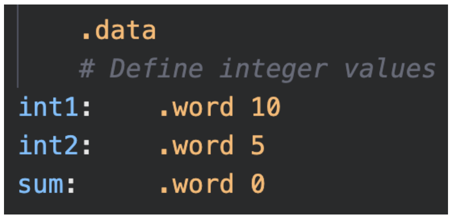
```
    .data
    # Define integer values
int1:    .word 10
int2:    .word 5
sum:     .doubleword 0
```

Error:



```
Warning in /Users/taarunianany a/Downloads/classes/comporg/lab8/Lab8.S line 23 colu
mn 10: RARS does not recognize the .doubleword directive.  Ignored.
```

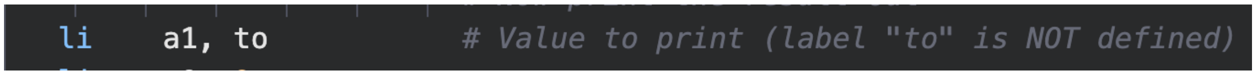
Fixed Code:



```
    .data
    # Define integer values
int1:    .word 10
int2:    .word 5
sum:     .word 0
```

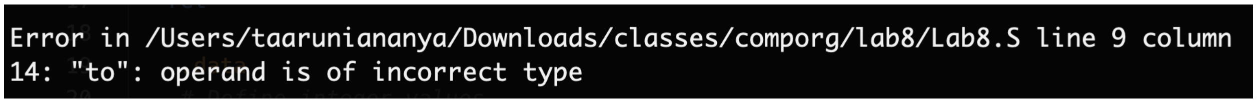
Error Five (to):

Original code:



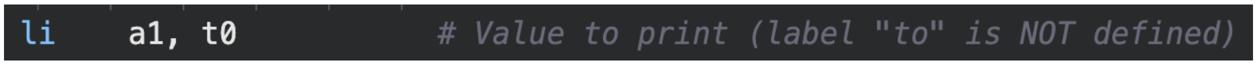
```
    li    a1, to            # Value to print (label "to" is NOT defined)
```

Error:



```
Error in /Users/taarunianananya/Downloads/classes/comporg/lab8/Lab8.S line 9 column
14: "to": operand is of incorrect type
```

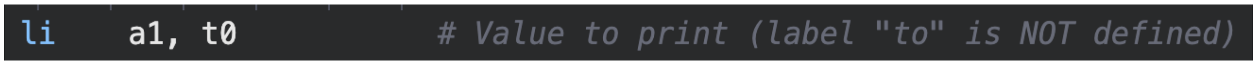
Fixed code:



```
li      a1, t0              # Value to print (label "to" is NOT defined)
```

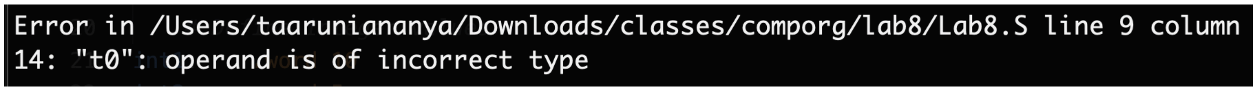
Error six (li):

Original code:



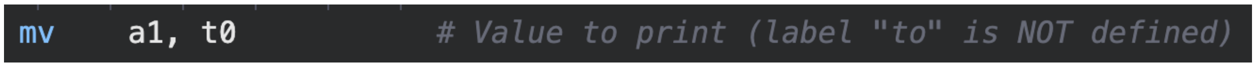
```
li      a1, t0              # Value to print (label "to" is NOT defined)
```

Error message:



```
Error in /Users/taarunianaya/Downloads/classes/comporg/lab8/Lab8.S line 9 column
14: "t0": operand is of incorrect type
```

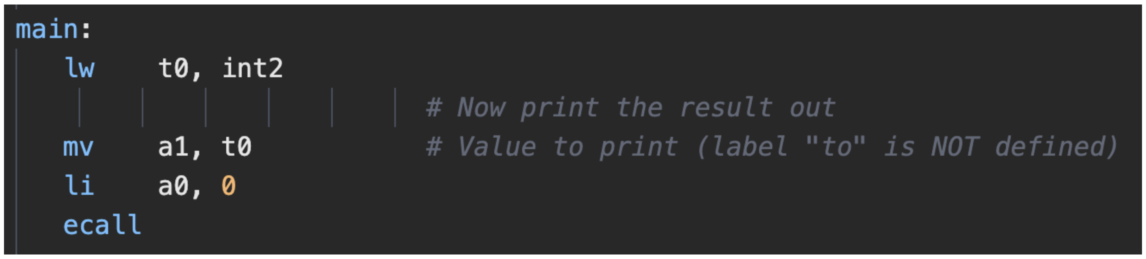
Fixed code:



```
mv      a1, t0              # Value to print (label "to" is NOT defined)
```

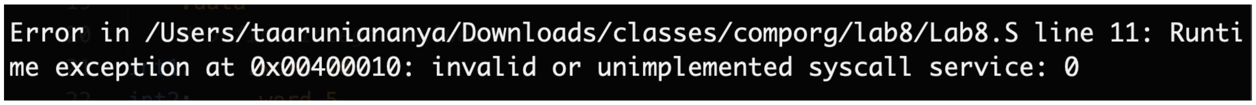
Error seven (ecall):

Original code



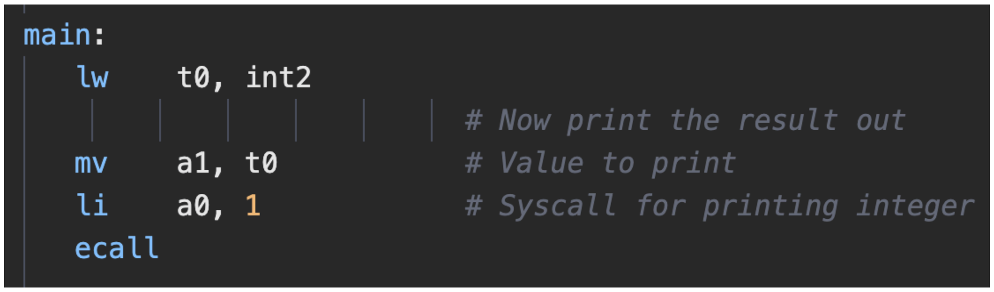
```
main:
    lw      t0, int2
                                # Now print the result out
    mv      a1, t0              # Value to print (label "to" is NOT defined)
    li      a0, 0
    ecall
```

Error message:



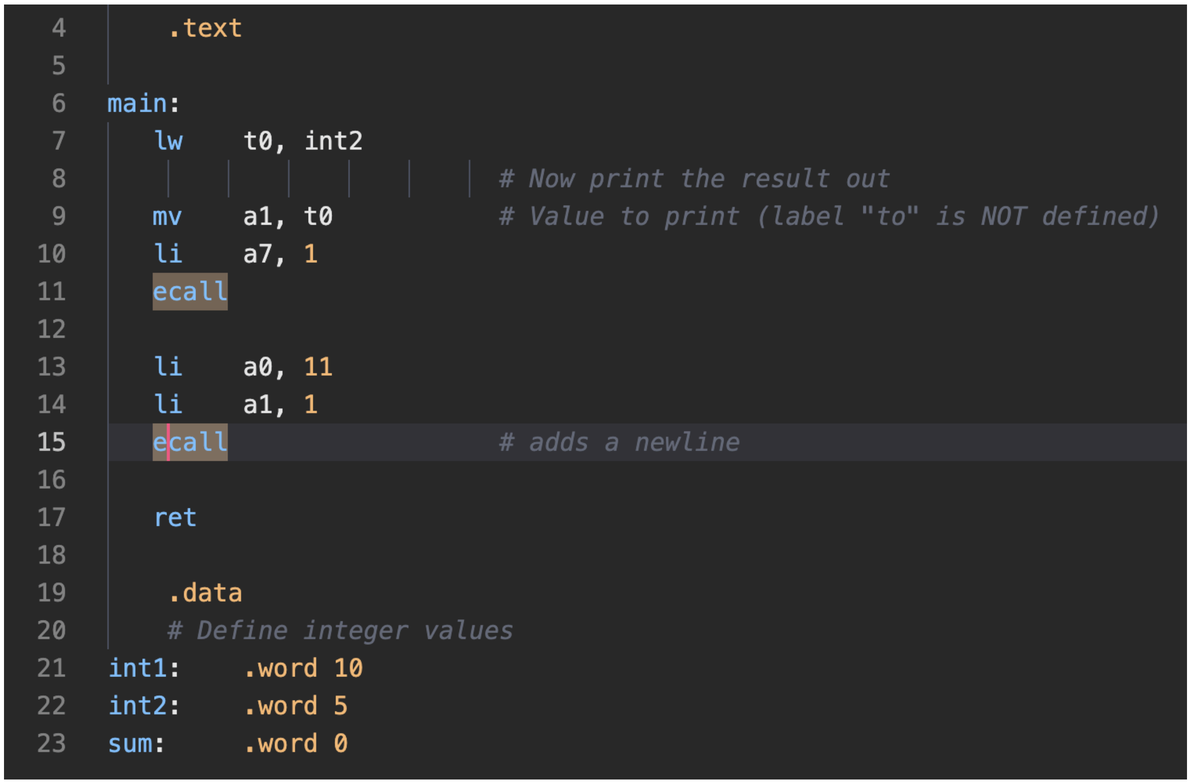
```
Error in /Users/taarunianaya/Downloads/classes/comporg/lab8/Lab8.S line 11: Runti
me exception at 0x00400010: invalid or unimplemented syscall service: 0
```

Fixed code:



```
main:
    lw     t0, int2
                                # Now print the result out
    mv     a1, t0               # Value to print
    li     a0, 1                # Syscall for printing integer
    ecall
```

Complete fixed code:



```
    .text

main:
    lw     t0, int2
                                # Now print the result out
    mv     a1, t0               # Value to print (label "to" is NOT defined)
    li     a7, 1
    ecall

    li     a0, 11
    li     a1, 1
    ecall                       # adds a newline

    ret

    .data
    # Define integer values
int1:   .word 10
int2:   .word 5
sum:    .word 0
```

Questions:

Part One:

1. **What is the effect of ret in the simulator in this program?**

ret is an instruction used to return after a function, like jumping back to a certain address. In this simulation, ret does not have a purpose since it doesn't seem to return to any certain address.

2. **What should ret be replaced with?**

   This program does not need to return to an address like a loop. As a result, having a call that can complete the function such as "exit" is best used here.

3. **You may have noticed that to get the program working, we had to figure out what the code was supposed to do. What could the previous programmer have done to make this clear?**

   The original code was very confusingly written, had multiple mistakes, and contained unnecessary code. The previous programmer could've used more comments, and even more detailed explanations to make what the program was supposed to do more clear.

4. **When we have multiple instances of semicolons, why would it be a bad idea to use a global find/replace operation?**

   Using semicolons is a way to make comments that do not alter or affect the main program written. By using global find/replace to change semicolons, this may accidentally replace/remove semicolons that are necessary to the program.

1. **Try changing the line**

```
 int2: .word 5
```

to

```
 int2: .double 5
```

then stepping through the program. What value is printed? What value does t0 get, and why?

When the line "`int2: .word 5`" is changed to "`int2: .double 5`", then stepped through the program, the resulting value printed is 0. t0 gets the value 0 as a result of the change from .word with 32-bit word to .double with 64 bits.

2. **How can you get rid of the breakpoints?**

   Pressing on the area or red circle next to the specific line of code that has a breakpoint is one way of getting rid of them.

3. **What registers changed their value?**

   One register that changed its value is t0 when it was changed from .word to .double. Another register that was changed was a1, where it was li at first then mv. The third register that was changed was a7, when it was changed from a0 to a7. Finally, a0 was also changed to 1 instead of 0.

4. **Did any of the registers change unexpectedly, and if so, why do you think that happened?**

t0 changed unexpectedly because the .word was changed to .double for int2, where it expected a double but was loaded because of lw (which usually takes on 32 bits, not 64).

5. **What are the minimum and maximum values that PC has for the program?**

   The minimum PC from the program is at the start with 0x00400000. The maximum value, as a result, is at the end of the program as 0x0040025 because each instruction but the last one is 4.

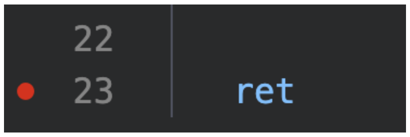
6. **As we advance from one instruction to the next, how much does PC's value change? Why?**

   Since each instruction in this program is Venus, RISC-V, there are 4 bytes each. As we advance from one instruction to the next, PV's value increases by 4 bytes, or 32-bits.

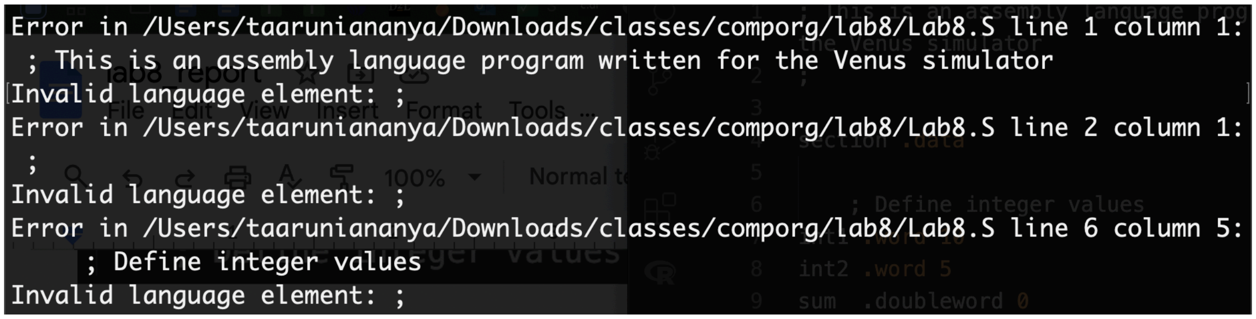
7. **Is there a direct relationship to the line number and PC? Why or why not?**

   No, there is not a direct relationship between line number and PC because there can be more empty lines without any instructions, meaning the PC value will not increase with an empty line–only for those with an instruction.

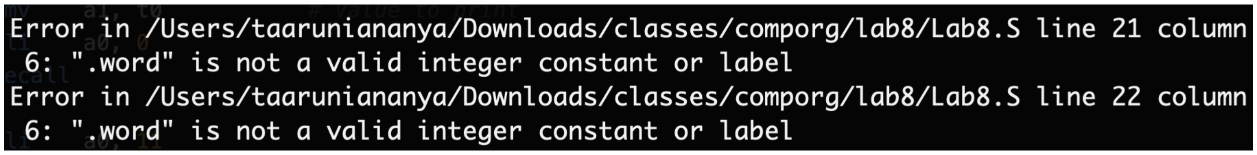
Key Code Observations



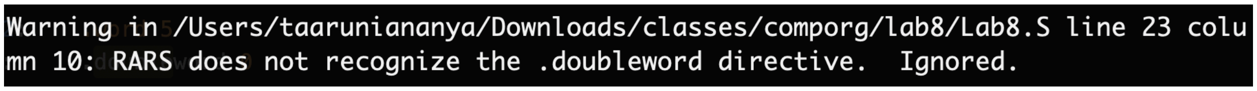
```
22
23        ret
```



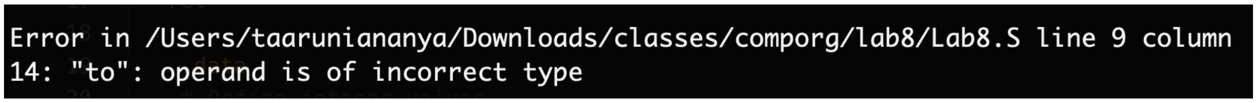
```
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 1 column 1:
 ; This is an assembly language program written for the Venus simulator
Invalid language element: ;
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 2 column 1:
 ;
Invalid language element: ;
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 6 column 5:
     ; Define integer values
Invalid language element: ;
```



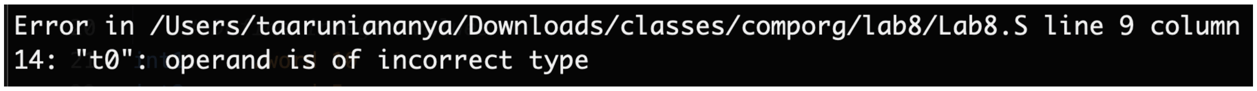
```
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 21 column
 6: ".word" is not a valid integer constant or label
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 22 column
 6: ".word" is not a valid integer constant or label
```



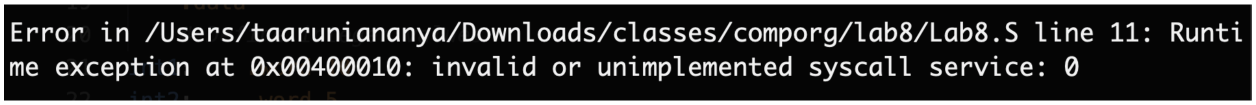
```
Warning in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 23 colu
mn 10: RARS does not recognize the .doubleword directive.  Ignored.
```



```
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 9 column
14: "to": operand is of incorrect type
```



```
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 9 column
14: "t0": operand is of incorrect type
```



```
Error in /Users/taaruniananya/Downloads/classes/comporg/lab8/Lab8.S line 11: Runti
me exception at 0x00400010: invalid or unimplemented syscall service: 0
```